Sissejuhatus infotehnoloogiasse

Ülevaade loengust

- Jätk eelmisest loengust: protsessor ja assembler
- Keeled: põhiideed, näited
- Cache ja mäluhierarhia

Hierarhia pistikutest progekeelteni

- Esimene: programmeerimismeetod: kaablid ja pistikud
- Teine: von Neumanni arhitektuur, programm mälus binaarkoodina:

```
01011101 01001011 01010101 11010101 10101001 ....
```

Lihtsam kirjutada hexas, ntx `4A FC 09 B2 ....`

- Kolmas: Esmane progekeel: assembler.

Üks masinakäsk: tüüpiliselt üks rida assembleriprogrammi

- Neljas: Harilik progekeel ehk nn kõrgkeel (fortran, basic, c, java, python jne jne).

Harilikud valemid, if-then-else jne, a la `x=2*y+sin(y);`

Ecki assembler

- This program counts. It starts by putting the number 1 into memory location 12, and then it adds one to the number in that location over and over, forever.

```
LOD-C 1
STO 12
LOD 12
INC
STO 12
JMP 2
```

```
          LOD-C 1      ; Set Count equal to 1
          STO Count
Loop:     LOD Count    ; Add 1 to Count
          INC
          STO Count
          JMP Loop     ; Jump back to start of loop

          @12
Count:    data ;  Location to be used for counting
```

Sumto MIPS-I (SGI spinoff) assembleris

- Argumendid registritesse $4 ja $8
- Resultaat registrisse $2

```
sumto:                      ; Register $4 on n
        li    $3, 0         ; Register $3 on summa
        li    $2, 0         ; Register $2 on i
        blt   $4, $0, L3    ; Kui n<0 mine L3
L5:     addu  $3, $3, $2    ; sum = sum + i
        addu  $2, $2, 1     ; i = i + 1
        ble   $2, $4, L5    ; Kui i<=n mine L5
L3:     move  $2, $3        ; Sum sisaldab resultaati.
        Jr    $31           ; Mine aadressile registris $31
```

Sumto ja Sun Sparc'i assembler

- Sparc saadab argumendid registrites %o0 kuni %o7 ja resultaadi %o0
- Instruktsioon peale hüpet tehakse alati

```
_sumto:                     ; Register %o0 on n.
        mov %o0,%g3         ; Salvesta n registrisse %g3.
        mov 0,  %o0         ; Register %o0 on nyyd sum.
        cmp %o0,%g3         ; Kui 0>n ...
        bg  L3              ; ... mine L3
        mov 0,  %g2         ; ,aga enne i=0.
        add %o0,%g2,%o0     ; sum = sum + i.
L5:     add %g2,1  ,%g2     ; i = i + 1.
        cmp %g2,%g3         ; Kui i<=n ...
        ble,a L5            ; ... mine L5
        add %o0,%g2,%o0     ; ,aga enne sum = sum + i.
L3:     retl                ; Valmis...
nop                         ; ,aga enne ära tee midagi!
```

Sumto ja Intel 386, 486, Pentium, ...

- 386 on vähe registreid, argument saadetakse hariliku mälu kaudu. Resultaat saadetakse registris %edx.

```
_sumto:
        pushl %ebp              ; Loome ''framepointer''-i
        movl  %esp,%ebp         ;
        movl  8(%ebp),%ecx      ; Võta n.
        xorl  %eax,%eax         ; sum = 0
        xorl  %edx,%edx         ; i = 0
        cmpl  %ecx,%eax         ; Kui i>n ...
        jg L3                   ; ... mine L3
        .align 2
L5:     addl  %edx,%eax         ; sum = sum + i
        incl  %edx              ; i = i+1
        cmpl  %ecx,%edx         ; Kui i<=n ...
        jle L5                  ; ... mine L5
L3:     leave                   ; Taasta ebp.
        ret                     ; Valmis!
```

CS Is About Programming

- **How to build systems.**
  - Better, faster, cheaper.
  - Reliable, maintainable, extensible.
- **Evaluating and comparing systems.**
  - Performance, behavior, security.
  - Compliance, usability.
- **What can and can’t be done.**
  - Algorithms and complexity.

Programming is linguistic

- **Programming is an explanatory activity.**
  - To yourself, now and in the future.
  - To whoever has to read your code.
  - To the compiler, which has to make it run.

- **Explanations require language.**
  - To state clearly what is going on.
  - To communicate ideas over space and time.

Programming languages are essential

- **Therefore languages are of the essence in computer science.**
  - Programming languages, in the familiar sense.
  - Description languages, design patterns, architecture languages, specification languages.

Kõrgkeeled

- **Automatiseerivad ja lihtsustavad** hulga “harilikke” protseduure, mida assembleris programmeerides vaja
- Ei anna assembleriga analoogilist kontrolli masina üle
- Kõrgkeeled on erineva abstraktsusastmega:
  - Masinalähedane ja ebamugav: Fortran, C (portaabel assembler)
  - Abstraktsem ja mugavam: Lisp, Ada, ML, Java, Python,
- **Peale programmeerimiskeelte on veel hulk muid keeli:**
  - Päringukeeled (SQL, RDQL, ...)
  - Kujunduskeeled (HTML, PS, ...)
  - Spetsifitseerimiskeeled (loogikakeeled, UML, ...)
  - ....

Kuidas keeles X kirjutatud programmi täidetakse?

Pea silmas, et: arvuti suudab täita ainult masinkoodis programme.

On olemas kaks põhivarianti keeles X programmi täitmiseks.

- **Kompileerimine**: masinkoodis programm nimega kompilaator teisendab keeles X programmi masinkoodfailiks Y. Seejärel täidetakse saadud masinkoodis programm Y. Näide: C.
- **Interpreteerimine**: masinkoodis programm nimega interpretaator loeb sisse X keeles faili ja asub seda rida-realt täitma. Näide: vana BASIC.

NB!

- Programmi interpreteerimine on ca 10-200 korda aeglasem, kui kompileeritud koodi täitmine.
- Põhimõtteliselt saaks igas keeles kirjutatud programme nii interpreteeritult täita kui kompileerida.
- Praktikas eelistatakse vahel interpreteerimist, vahel kompileerimist.

Kuidas keeles X kirjutatud programmi täidetakse?

Kompromissvariante:

- Kompilaator kompileerib X faili **vahekoodiks** Y, seejärel interpreteeritakse vahekoodi Y (Python, Java).

- Interpretaator interpreteerib vahekoodi Y, kuid **kompileerib töö ajal osa Y-st masinkoodiks**, mida seejärel täidab (Java) nn just-in-time compilation ehk **JIT**.

Kompileeritava programmi valmimine

Olgu meil (näiteks C keeles) failid main.c ja swap.c

Teeme gcc main.c swap.c -o minuprogramm

Kompilaator (näiteks gcc) teeb järjest mitut eri asja:

- **Kompileerimine**
  - Kompilaator teeb neist assemblerikeelsed ajutised failid
  - Kompilaator teeb assemblerfailidest masinkood+sümbolinfo failid
- **Linkimine**
  - Linkur otsib kokku vajalikud olemasolevad failid osa sümbolinfo seostamiseks päris koodi-viidetega

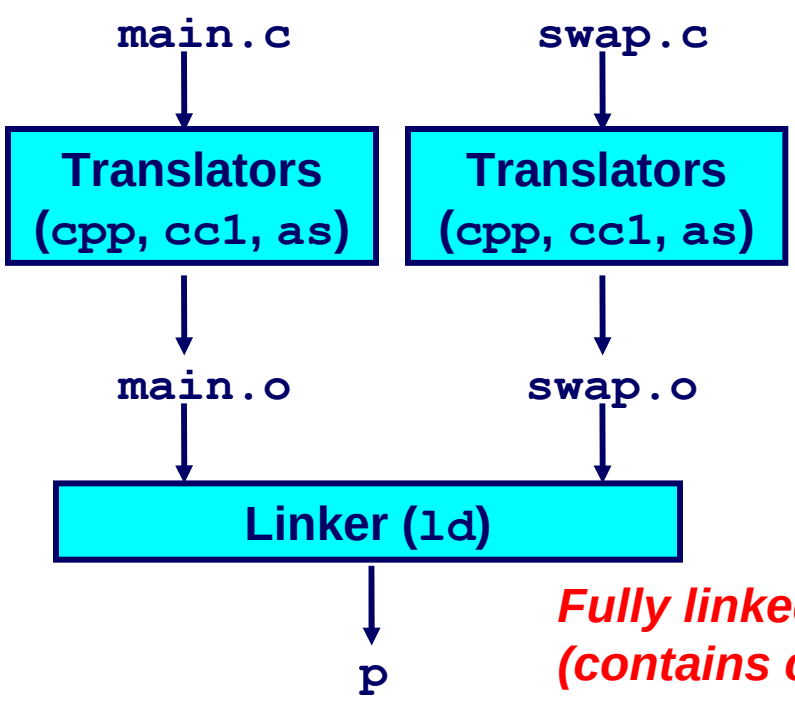
Käivitame saadud programmifaili minuprogramm:

- Opsüsteemi **loader** otsib lisaks vajalikud olemasolevad failid osa sümbolinfo seostamiseks päris koodi-viidetega
- Saadud kogum paigutatakse mällu, tehakse opsüsteemi infoblokk tema jaoks (protsess) ja kogum käivitatakse

(CMU Bryant & Hallaron course) Static Linking

Programs are translated and linked using a *compiler driver*:

- `unix> gcc -O2 -g -o p main.c swap.c`
- `unix> ./p`

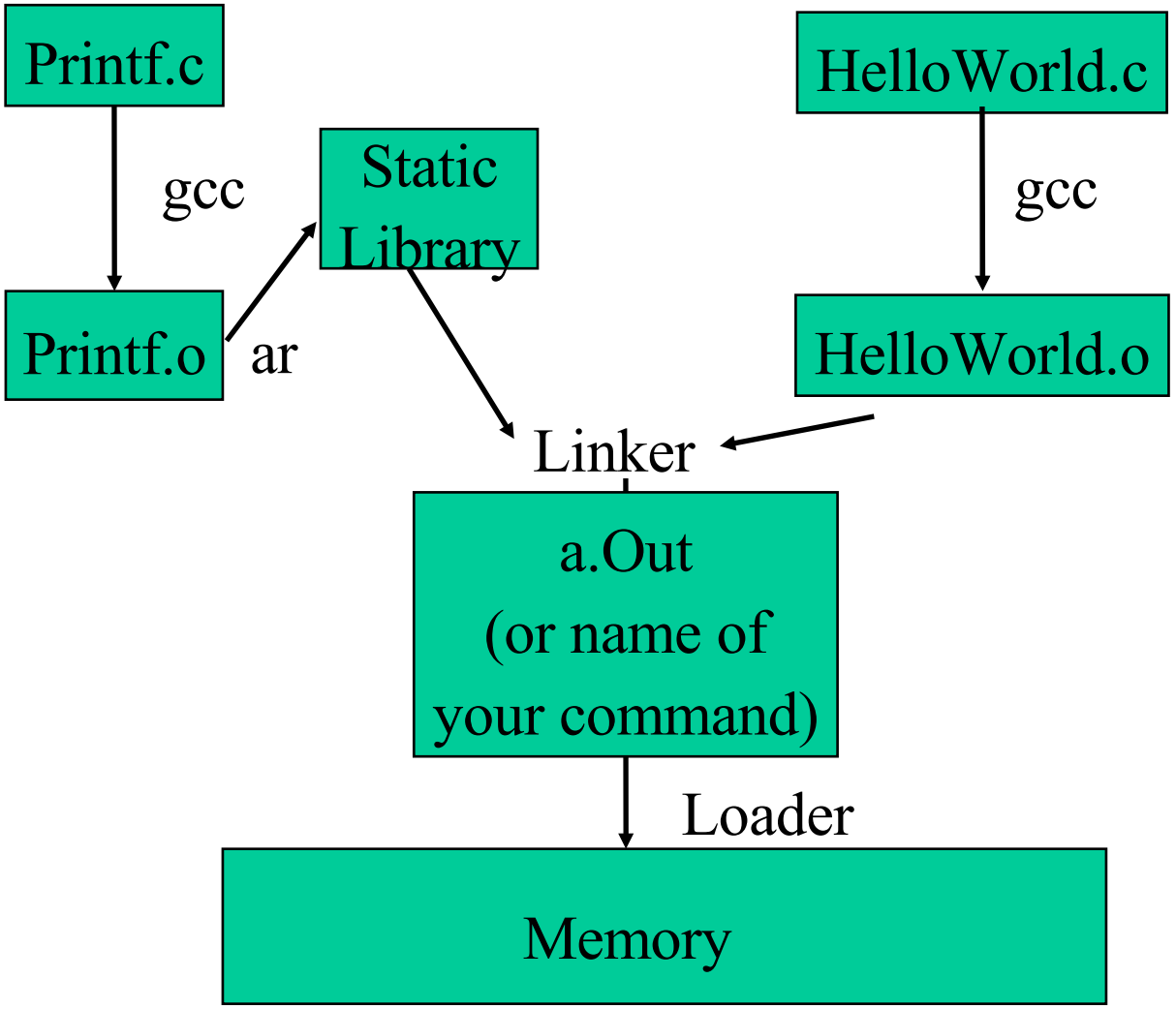


Source files

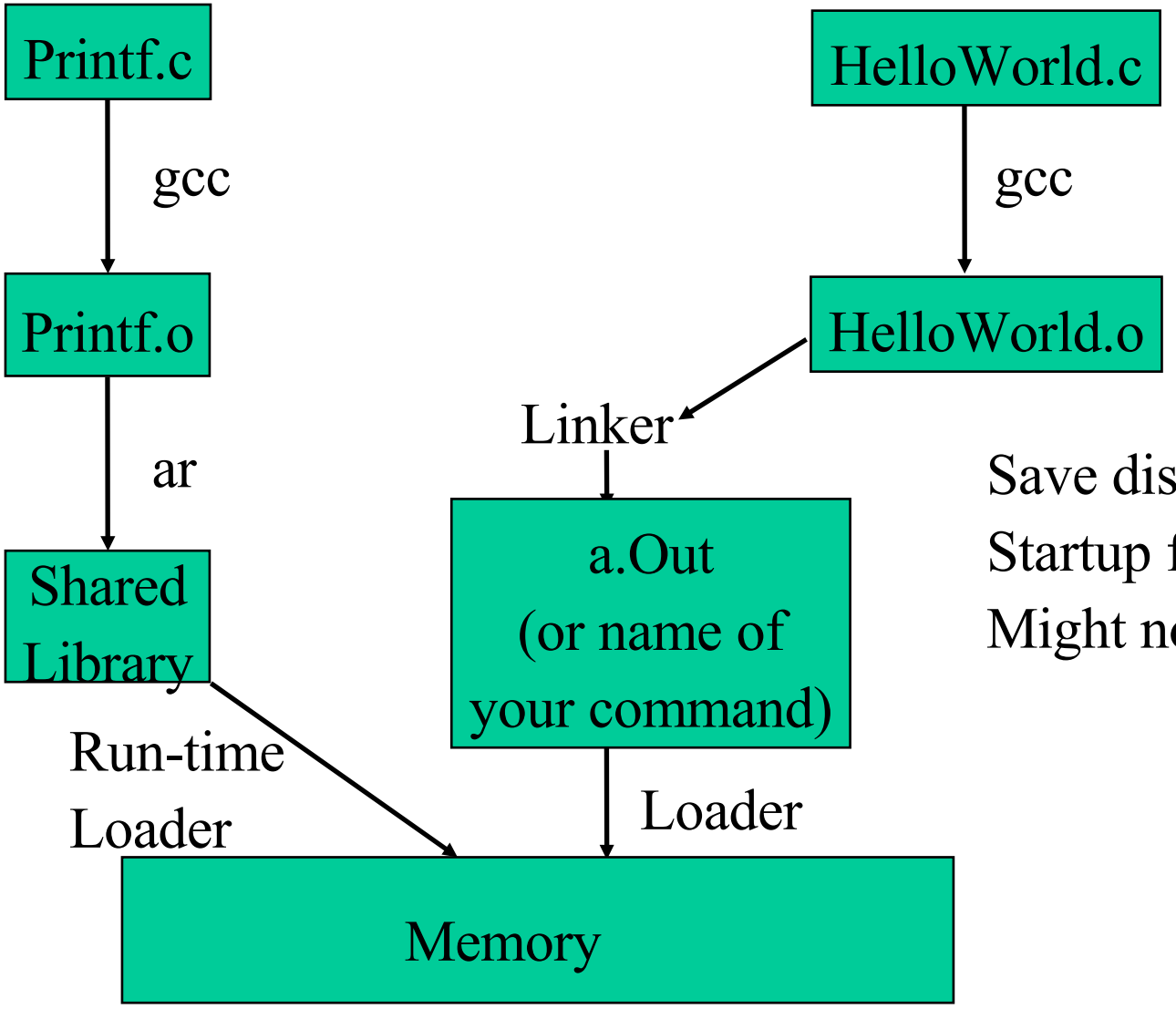
Separately compiled relocatable object files

Fully linked executable object file (contains code and data for all functions defined in main.c and swap.c

(CMU B&H ..) Static Linking and Loading



(CMU B&H ..) Run-time Linking/Loading



Save disk space.
Startup faster.
Might not need all.

Keelte erisused: kolm põhiasja

- **Süntaks** (kuidas kirjutatakse näiteks **if .. then .. else** ühes või teises keeles)
- **Semantika** ehk tähendus (mida õigesti kirjutatud programm tegelikult siis teeb)
- **Teegid (libraries)** (millised **valmisprogrammijupid** on selle keele jaoks kergesti kättesaadavad või kohe kaasa pandud)

Keeled: tüüpilised asjad, mida pea iga keel pakub

- **Primitiivsed andmetüübid:**
  - int, char etc (näiteks: 1 ja –3 on int-id, ‘c’ ja ‘a’ on char-id)
  - string (näiteks “aaa123bb”)
  - massiiv (näiteks a[1]=2; a[2]=20; a[3]=15; y=2; x=a[y] +a[1]+3;)
- **Avaldised:**
  - näiteks x = (y*2) – (5+x);
- **Elementaarsed juhtkonstruktsioonid:**
  - valik: if ... then ... else
  - tsükkel: while(x<10) x=x+1;
- **Funktsioonid:**
  - defineerime: int kuup(int x) { return x * x * x}
  - kasutame: x = kuup(1+kuup(3))+kuup(y);
  - kasutame rekursiivselt:

int fact(int x) { if (x<=0) return 1; else return x*(fact(x-1)); }

Keeled: näited lisavõimalustest eri keeltes

- Kiired bitioperatsioonid, otsepöördumine mälu kallale: C
- Keerulisemad andmetüübid: listid, hash tabelid jne: Lisp, Scheme, Python
- Erikonstruktsioonid stringitöötluseks: Perl, PHP

- Objektid: C++, Java, C#, Python, Lisp
- Moodulid (enamasti ühendatud objektidega): C++, Java, C#
- Veatöötluse konstruktsioonid (exceptions): Python, Java, C#

- Prahikoristus: kasutamata andmed visatakse välja (Java, Python, Lisp, ...)
- Sisse-ehitatud tugi paralleelprogrammide jaoks: Java, C#
- Reaalaja-erivahendid: Ada

- “Templates” (programm tulemuse sees): PHP, JSP, Pyml
- Uute programmide konstrueerimine töö käigus: Lisp, Scheme
- Loogikareeglid: Prolog
- “laisk” viis funktsioone arvutada: Miranda, Hope, Haskell
- Pattern matching (viis funktsioone defineerida): ML, Haskell
- **jne...**

Fortran: summeeri arve 0...n

- FORTRAN

```
      INTEGER FUNCTI0N sumto(n)
      isum = 0
      DO i 10 = 0,n
      isum = isum + i
10    CONTINUE
      sumto = isum
      RETURN
      END
```

COBOL: summeeri arve 0...n

- COBOL

```
PROCEDURE SUMTO USING N, Answer.
Begin.
  PERFORM VARYING LoopCount FROM 0 BY 1
    UNTIL LoopCount GREATER THAN N
      MULTIPLY Answer BY LoopCount GIVING Answer.
  END-PERFORM.
  EXIT PROGRAM.
```

LISP: summeeri arve 0...n

- LISP

```
(defun sumto (n)
   (if (= 0 n)
       0
       (+ n (sumto (- n 1))) ))
```

Sumto ja C

- C (ja C++ ja Java ja C#)

```
int sumto(int n) {
  int i,sum = 0;
  for(i=0; i<=n; i=i+1)
    sum = sum + i;
  return sum;
}
```

Sumto ja Modula-2

- Modula-2

```
PROCEDURE sumto(n:INTEGER):INTEGER;
VAR sum,i:INTEGER;
BEGIN
      sum:=0;
      FOR i:=0 TO n DO
        sum:=sum+i
      END;
      RETURN sum
END sumto;
```

Sumto ja Ada

- Ada

```
function sumto(n: in INTEGER) return INTEGER is
    sum : INTEGER := 0;
begin
    for i in 0..n loop
       sum := sum + i;
    end loop;
return sum;
```

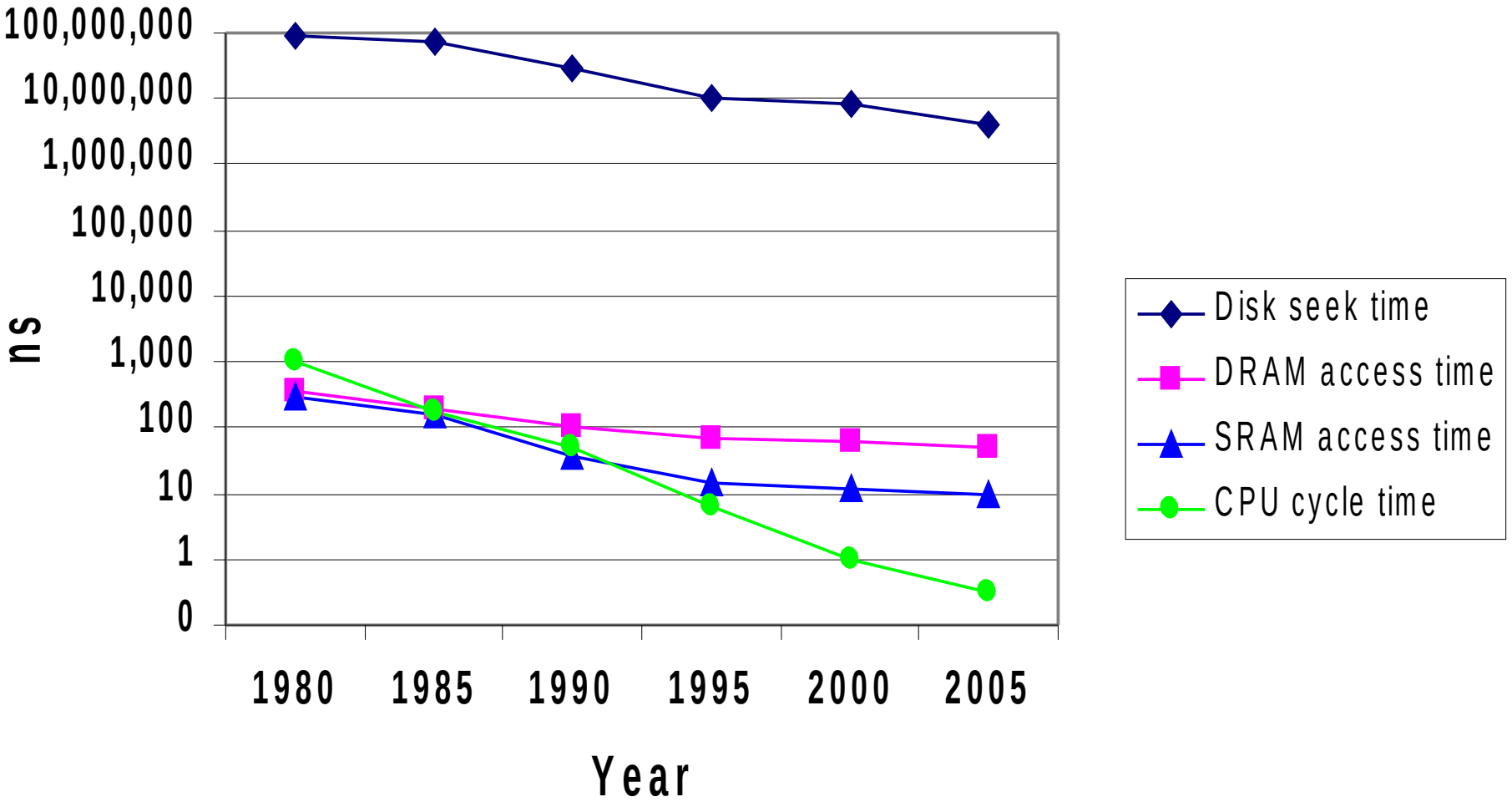
Sumto ja Python

- Python

```
def sumto(n):
  sum=0
  for i in range(n+1):
    sum=sum+i
  return sum
```

(CMU B&H ..) The CPU-Memory Gap

The gap widens between DRAM, disk, and CPU speeds.



Cache ja mälu hierarhia